A picture containing arrow

Description automatically generated

**ENGO 531 – Advanced Photogrammetric and Ranging Techniques**

**Lab Report #1**

**Course Instructor:** Dr Derek Lichti

**Teaching Assistant:** Sandra Simenova

**Due Date:** 10/12/21

**Date of Submission:** 10/12/21

**Student Name:** Jan Erik Naess (30068695)

Table of Contents

[Table of Figures 3](#_Toc84978239)

[Introduction 4](#_Toc84978240)

[Documented Source Code 4](#_Toc84978241)

[Main Classes 4](#_Toc84978242)

[Assistive Classes 4](#_Toc84978243)

[Function Documentation 4](#_Toc84978244)

[General Documentation 5](#_Toc84978245)

[Formatted Bundle Adjustment Output File (Validation) 6](#_Toc84978246)

[Verification Report 7](#_Toc84978247)

[Adjustment Configuration Input File 10](#_Toc84978248)

[Short Answer to Question 10](#_Toc84978249)

[Input File Spacing 10](#_Toc84978250)

[Poor Tie Point Count 11](#_Toc84978251)

[High Correlation 11](#_Toc84978252)

[Blunders 11](#_Toc84978253)

[Discussion 12](#_Toc84978254)

[Unit Conversion Importance 12](#_Toc84978255)

[Conversion Criteria 12](#_Toc84978256)

[Automated output of results and matrices 12](#_Toc84978257)

[References 13](#_Toc84978258)

[Appendices 13](#_Toc84978259)

[Cl Values ( 13](#_Toc84978260)

[Cr values 13](#_Toc84978261)

[Final Output File 13](#_Toc84978262)

[Github Repository 22](#_Toc84978263)

[https://github.com/jnaess/ENGO531.git 22](#_Toc84978264)

[Github Commits 22](#_Toc84978265)

[Tools.py 24](#_Toc84978266)

[Net.py 25](#_Toc84978267)

[LeastSquares.py 33](#_Toc84978268)

[PostAdjustmentTester.py 35](#_Toc84978269)

[Design\_o.py 40](#_Toc84978270)

[Tables.py 45](#_Toc84978271)

[FileReader.py 46](#_Toc84978272)

[Bundle.py 49](#_Toc84978273)

# Table of Figures

[Figure 1: Final Output File (sample) 6](#_Toc84972373)

[Figure 2: Global A-Posteriori Variance Factor Test 7](#_Toc84972374)

[Figure 3: Significance of Estimated Parameters (sample) 8](#_Toc84972375)

[Figure 4: Blunder Test (sample) 9](#_Toc84972376)

[Figure 5: Adjusted Observations (m and radians) **Error! Bookmark not defined.**](#_Toc84972377)

[Figure 6: Covariance Matrix of Adjusted Observations (see appendix for numerical values) **Error! Bookmark not defined.**](#_Toc84972378)

[Figure 7: Observation Residuals (m and radians) **Error! Bookmark not defined.**](#_Toc84972379)

[Figure 8: Covariance Matrix of Residuals (See appendix for numerical values) **Error! Bookmark not defined.**](#_Toc84972380)

[Figure 9: A-Posteriori Variance Factor(unitless) **Error! Bookmark not defined.**](#_Toc84972381)

# Introduction

This lab was constructed to begin developing a full-stack bundle adjustment. Python was chosen because of its modularity capabilities, ability to read in data and store with DataFrames, and for its general ease of use. Software was developed in an Object Oriented Manner, currently housing 8+ classes and dozens of functions. This high level of modularity allowed for efficient debugging and upgrading of code. Data was validated with DataFrames that allow the user to call the desired information with one line of code. Data was verified with post adjustment tests specially formatted for a Bundle Adjustment. All code was sufficiently documented both inline and via descriptive function headers. An input file was built in to read and write the components that were specific to this dataset. Several issues with the data were found and addressed at the end of this lab report.

# Documented Source Code

## Main Classes

Software was developed within Python using common libraries such as math, numpy and pandas. Data was visualized using matplotlib and was processed with four in-house classes. The “Network” class conducted the overarching LSA and generates final output values for the assistive classes. The one functional model class, “Design\_o” which generates all colinear matrices before the least-square adjustment and updates the design matrices for the LSA. The “LSA” super class was inherited by all classes and contained metrics such as and helped with sorting of columns to correctly input and update important matrices.

## Assistive Classes

The “PostAdjustmentTester” class was made to autogenerate the results displayed in the verification report and leverages the “Net” class and “Tools” class to generate and output/save results. The “Tools” and “Tables” classes were made to break down functionality that was needed within other classes and allow then to be inherited for functionality such as outputting files or splitting DataFrames in unique ways.

## Function Documentation

All functions were documented with a standard Description, Input and Output portion. The description consisted of the use-case for that function and included any notes about potential changes or scope constraints to the function’s use. The input contained all variables that needed to be either read in or initialized before calling the function. The output contained all files that were either returned, updated, or initialized by calling this function. See below for a sample function documentation that was used within the LeastSquares class.

Text

Description automatically generated

Figure 1: Sample Function Documentation

## General Documentation

The rough goal was to have a comment for nearly every line of code that was written. In general, comments were recorded for each minor piece of functionality within a function. This assisted greatly in debugging and building upon old code.

# Formatted Bundle Adjustment Output File (Validation)

The software package developed leverages pandas DataFrames to output whichever values are desired. A function to output the DataFrame with a file name is located within the Tools class and autonomously creates a folder called “Files” and outputs the DataFrame as a csv. Overall, the data seems to have converged to a reasonable standard deviation. It can be seen that most of the angles have a low angular standard deviation and that points locations for the images are within .5mm of any of the three axises. Control points largely kept their values and were not greatly influenced by the adjustment which was a desired result. Image points had standard deviations that stayed under 2mm which is a reasonable result given some of the conditions such as only one tie point for one of the images. Below is a clipping of the outputted file which contains the standard deviation for the final points.

Table

Description automatically generated

Figure 2: Final Output File (sample)

DataFrames or matrices available for autonomous outputting include but are not limited to: all input files (con, ext, int, out, pho, tie), combined input files (obj), all final matrices (Cl, Cr, x\_hat, w\_0, S\_hat, etc.), all initial matrices (x\_0, obs, l\_0, etc.) and any of the PostAdjustmentTester dataframes.

# Verification Report

The PostAdjustmentTester class was generated to verify the results of the least square adjustment. Overall, the potential issue with the models used was that the additional control points as fake observations were not input due to time constraints. This resulted in an a-posteriori factor that was slightly above the example which generated a value of 7, while this generated a value of 10.5 as seen below.

Graphical user interface, text

Description automatically generated

Figure 31: A-Posteriori Variance Factor (unitless)

This was compared to an a-priori of 1 in the global test. Because of the high level of redundancy, the test required a better set of functional models in order to make full use of the large amount of redundancy. The test and respective analysis may be seen below.



Figure 42: Global A-Posteriori Variance Factor Test

The a-posteriori variance factor test failed indicating that there may have been errors or poorly chosen math models. The failure of this test is a strong enough indication that other components should be tested but is not a guarantee that there are any major issues.

Table

Description automatically generated

Figure 5: Significance of Estimated Parameters (sample)

The significance of all parameters was then assessed. This checked to see if there was evidence that the parameter had statistical significance to be used within the model and for future applications. The test resulted in an overwhelming yes, indicating that the parameters had reached desirable levels of precision given the functional models and quality of the dataset.



Figure 6: Goodness-of-fit test

Next the goodness-of-fit test was conducted. The failure of this test indicated that there may have either been outliers or portions of the data that were of relatively low quality. One of these points may be been because there was only one tie point in image four which greatly reduced its quality.

Table

Description automatically generated

Figure 7: Blunder Test (sample)

A blunder test was conducted on all observations in order to check whether they may be outliers. This test was conducted to the 99% confidence interval and resulted in several observations being considered outliers. However, the data proved largely free of outliers, and where there was one indicated, there was enough redundancy to compensate for it.

# Adjustment Configuration Input File

Key values for the Network Adjustment were input into the Input.txt file. This file consisted of only two row, one for the header names and the second for the respective values that would be read in. These values were used by both the “Bundle” class and “Design\_o” class to initialize their variables. Below is the input DataFrame that is read in and accessed.



Figure 8: Input.txt DataFrame

In order to add new values all that the user needs to do is ass the variable name and the assign its to the class once it is read in. Below is the format of the file, both white spaces and commas may be used to separate variables and numbers.



Figure 3: Input.txt composition

# Short Answer to Question

*The quality of this dataset is such that the bundle adjustment will converge within a few iterations, depending on the tolerances used. However, the dataset contains at least three “problems”. Identify and briefly explain these problems*

## Input File Spacing

A plethora of issues were found in the consistency of the input file separation techniques. Separation between values varied both between files and within each file themselves. Some examples included extra tabs after completion of a line, using whitespaces instead of tabs and using varying lengths of white spaces.

Pandas’ read\_csv() function was leveraged to read in these files regardless of their poor formatting. By setting the delim\_whitespace parameter to be True, all files were read in based on commas, tabs, and all variations of spacing. Below is the function and parameter in that was called to read in the .pho observation file.



Figure 10: pd.read\_csv function

## Poor Tie Point Count

Image four included only one tie point. In order to better stich together the images that were taken, it is common practice to have a minimum of four or five tie points. However, negative effects of this were mitigated by using control points between images. That being said, the issue with using control points is that they offer very little room to move because of their heavy weights. This results in the images have a low level of willingness to readjust to possibly better positions.

## High Correlation

Correlations between the EOP’s were relatively high which indicated that the final positions may have systematic errors. Correlation may be reduced if a different model is used or if there is a larger amount of redundancy. Therefore, the easiest fix would be to collect more data or to use a second camera to collect datapoints of tie points and control points from a different set of parameters so that there would be less reliance on the IOP’s of only one camera. As always, more redundancy is better.

## Blunders

Both the example output, and the software’s output detected various blunders at the 99% confidence level. Because of the large amount of redundancy, it would be wise to redo the LSA, taking the largest blunder out each time until no blunders were detected.

# Discussion

## Unit Conversion Importance

It is common practice in surveying to provide angular precision in arc seconds and distance precision in whole numbers (2mm instead of .002m). There are two important adjustments that must made to errors in order to integrate them into a programable least-squares-adjustment. The first adjustment is to the angular error, which must be converted into decimal degrees, and is often then converted from degrees to radians. Radians is preferred because most code libraries by default read and return angle measurements in radians. The second adjustment that should be made is in the conversion of non-meter errors into decimal millimeters. This is because measurements and positioning are normally provided in meters. If they are provided in a different unit, then all distance and x, y, z coordinates and errors should at the bare minimum be uniform. Otherwise, disproportional adjustments will occur.

The importance of conversion from LHC to RHC should also be noted. In the future it would be wise to write down which coordinate system functional models output as so that respective conversions may be made without unnecessary debugging.

## Conversion Criteria

It is a commonly accepted practice to have the conversion criteria of an LSA be based off of the parameter’s standard deviations. This means that Cx is computed and then the diagonal elements are taken out and square rooted to see their standard deviation. Once all standard deviations are below one-half of the observation’s standard deviations then it is often acceptable to end convergence. Because the functional models used were well suited for this application, a simpler convergence criterion was used. This LSA was programmed to meet convergence at .0001 mm: of which was converged to after the fourth iteration.

## Automated output of results and matrices

Least-squares-adjustments often use repeatable statistical tests and desire similar formats of outputted results. Additional time was invested in this lab to create several classes for performing fully autonomous analysis and matrix figure generation. The “Tools” class was created to automate visualization of importance matrices. The “PostAdjustmentTester” class was created to leverage final matrices outputs and conduct statistical tests on them. Lastly, the “Tables” class was to consolidate all statistic table values in an easily accessible and formattable location so that statistical tests could be easily conducted and automated.

# References

Gao, Y. (2021). *Lab1 - Instructions.* Retrieved from D2L: <https://d2l.ucalgary.ca/d2l/le/content/399854/viewContent/4877097/View>

El-Sheimy, N. (2021). *Review of least squares (parametric).* Retrieved from D2L: <https://d2l.ucalgary.ca/d2l/le/content/399854/viewContent/4843398/View>

Detchev, I. (2020). *Examples for Post-Adjustment Tests.* [PDF]

# Appendices

## Cl Values (

matrix([[ 6.74823301e-07, 3.34208092e-08, 1.38005978e-07, ...,

-2.75125218e-09, -4.51771739e-09, -1.20103519e-08],

[ 3.34208092e-08, 9.11841435e-07, 6.22579638e-08, ...,

1.26568697e-09, 3.91013344e-09, -2.14700648e-09],

[ 1.38005978e-07, 6.22579638e-08, 5.44082447e-07, ...,

-8.20836564e-09, -8.80541832e-09, -1.39891662e-08],

...,

[-2.75125218e-09, 1.26568697e-09, -8.20836564e-09, ...,

1.23082211e-06, 3.26215153e-07, 2.95061082e-07],

[-4.51771739e-09, 3.91013344e-09, -8.80541832e-09, ...,

3.26215153e-07, 1.43532329e-06, 3.61296839e-07],

[-1.20103519e-08, -2.14700648e-09, -1.39891662e-08, ...,

2.95061082e-07, 3.61296839e-07, 8.71448720e-07]])

## Cr values

matrix([[ 1.31893339e-03, -3.34208092e-08, -1.38005978e-07, ...,

2.75125218e-09, 4.51771739e-09, 1.20103519e-08],

[-3.34208092e-08, 1.31869637e-03, -6.22579638e-08, ...,

-1.26568697e-09, -3.91013344e-09, 2.14700648e-09],

[-1.38005978e-07, -6.22579638e-08, 1.31906413e-03, ...,

8.20836564e-09, 8.80541832e-09, 1.39891662e-08],

...,

[ 2.75125218e-09, -1.26568697e-09, 8.20836564e-09, ...,

1.31837739e-03, -3.26215153e-07, -2.95061082e-07],

[ 4.51771739e-09, -3.91013344e-09, 8.80541832e-09, ...,

-3.26215153e-07, 1.31817289e-03, -3.61296839e-07],

[ 1.20103519e-08, 2.14700648e-09, 1.39891662e-08, ...,

-2.95061082e-07, -3.61296839e-07, 1.31873676e-03]])

## Final Output File

,Unknown,Final Value (mm or rad),Value Standard Deviation (mm or rad)

0,1Xcj,-746.9548816473772,0.47234639090886116

1,1Ycj,-268.59453481058773,0.5463283331279489

2,1Zcj,113.70221338066949,0.7857857785265298

3,1w,1.6172539672138702,0.00029455035566097027

4,1o,-0.34500465628088756,0.00019287068297236745

5,1k,-0.05286714511601137,0.00015720041486024032

6,5Xcj,2614.1379095145135,0.5830226975124676

7,5Ycj,89.3585670723902,0.5683140735063945

8,5Zcj,110.03459675304649,0.7267965198349986

9,5w,1.5648278842545995,0.0003066707722978768

10,5o,0.6643685871286442,0.00023815489294351521

11,5k,-0.03125266606156934,0.00017181705668870259

12,8Xcj,-122.24395610749683,0.5916583459119032

13,8Ycj,-665.8478573599588,0.3655268136354254

14,8Zcj,61.10353279014381,0.7875344443864282

15,8w,1.4846016896456586,0.000270344414528684

16,8o,-0.20426343919457873,0.000213760265605699

17,8k,1.5486101077566996,0.0001480899797660257

18,9Xcj,537.074443237871,1.309577044172131

19,9Ycj,-633.9989119913413,0.6502007723911171

20,9Zcj,57.64706144648279,1.8092673258249081

21,9w,1.4815126212776706,0.0005441070167349298

22,9o,-0.09309534507325597,0.00039695156806900404

23,9k,1.5768299188781774,0.00020041293178162085

24,24Xi,-0.09166306174906819,0.009999165806853212

25,24Yi,1700.0022530724082,0.009999744523501405

26,24Zi,100.12851688797763,0.009998958765750757

27,71Xi,2200.0917465010602,0.009999327705273975

28,71Yi,2499.980564030343,0.009999771643717221

29,71Zi,-1200.0267266151197,0.00999943648980049

30,85Xi,2762.3830142962056,9.967323342003434

31,85Yi,1972.3483630592066,10.437907505927175

32,85Zi,470.6209001182112,1.9562866541856834

33,89Xi,2770.6058521373275,0.583547210332845

34,89Yi,1973.8472563141615,1.3487743807474804

35,89Zi,-228.29113708077492,0.6733761503442887

36,132Xi,1799.9588938710501,0.009999620211336296

37,132Yi,2999.988907736204,0.009999900140227568

38,132Zi,-800.0035974025712,0.009999613130439637

39,133Xi,2300.066148951539,0.009999250512698173

40,133Yi,2400.016969910048,0.009999827776666153

41,133Zi,-599.9254192525424,0.009999361461033926

42,A5Xi,46.99837427920278,0.009999120048456687

43,A5Yi,1787.9902454560681,0.009999290966118436

44,A5Zi,1982.0075490227478,0.009999027692961621

45,B1Xi,1646.0055155105613,0.00999940593587011

46,B1Yi,3051.9992635012086,0.009999832623166916

47,B1Zi,1258.003800031549,0.00999935381676519

48,B2Xi,2225.0011853992123,0.009999023660392013

49,B2Yi,2489.9992496611776,0.009999634025090777

50,B2Zi,1254.002153316236,0.009999055573122606

51,B8Xi,1667.001954568516,0.009999413958788015

52,B8Yi,3038.0010785921386,0.009999880301437688

53,B8Zi,488.9977624099092,0.00999935202246111

54,B9Xi,2214.993602906973,0.009999072309933504

55,B9Yi,2506.003104893984,0.009999756322524233

56,B9Zi,460.9917098422195,0.009999071248749724

57,A32Xi,-759.9860751912686,0.009997868925721095

58,A32Yi,987.9960298743194,0.009999218051006283

59,A32Zi,-547.0244577726111,0.009998415122966349

60,A33Xi,-181.99972579399545,0.009999022253982375

61,A33Yi,1554.9982308198673,0.00999963828361255

62,A33Zi,-541.007779812752,0.009998836220797102

63,A34Xi,391.9962337582829,0.009999164042695146

64,A34Yi,2121.9991246098284,0.009999764524707664

65,A34Zi,-547.0063228444194,0.009998998305791495

66,A35Xi,975.9952435053966,0.009999391391597558

67,A35Yi,2702.99940863817,0.009999851973240867

68,A35Zi,-558.0032366302449,0.009999283826059475

69,BS8Xi,2558.992734660378,0.009998531318041244

70,BS8Yi,2177.006027587869,0.009999563229771004

71,BS8Zi,700.9885663325562,0.009998808188310155

72,AS10Xi,236.0011440434026,0.009998973132279227

73,AS10Yi,1981.9946898772273,0.00999937167035915

74,AS10Zi,1751.0026138316653,0.00999881027459042

75,AS12Xi,-380.9936993129382,0.009998680626491431

76,AS12Yi,1372.997845180642,0.009999098090579273

77,AS12Zi,1463.9961271380248,0.009998677354957453

78,AS31Xi,-867.9814414476119,0.009999002884843221

79,AS31Yi,889.0171677406566,0.00999901307628166

80,AS31Zi,-1221.9955368658143,0.009999232563890126

81,AS32Xi,839.9930657617855,0.009999520319999625

82,AS32Yi,2580.9970830693983,0.009999846684807507

83,AS32Zi,-1196.0049173972925,0.00999947402669792

84,BS11Xi,1585.0021490529202,0.00999944964277421

85,BS11Yi,3128.0013926652487,0.009999898663210416

86,BS11Zi,248.9967482820198,0.009999390341903034

87,BS17Xi,1766.9981257155125,0.009999619982285919

88,BS17Yi,2956.0014345122127,0.009999895773972802

89,BS17Zi,-795.99884511165,0.00999960634427488

90,BS20Xi,2561.985486163508,0.009998799042748295

91,BS20Yi,2178.999164487887,0.009999652551347491

92,BS20Zi,-1033.0072388263068,0.009999216307153521

93,BVS5Xi,2444.0005729879563,0.009999610636187754

94,BVS5Yi,2270.9917172963646,0.009999579394370309

95,BVS5Zi,1610.0129047642085,0.009999414851668697

96,BVS6Xi,2212.0048298027004,0.0099990231287243

97,BVS6Yi,2497.9946101469054,0.009999559227744117

98,BVS6Zi,1613.0104389518888,0.009999080443748741

99,AVS50Xi,63.007814043732544,0.009998809042843268

100,AVS50Yi,1798.999737220244,0.009999263110786276

101,AVS50Zi,1627.9970328220686,0.009998649313436102

102,AVS51Xi,345.00026806027086,0.009999291781789845

103,AVS51Yi,2074.9967768488004,0.009999618850171674

104,AVS51Zi,1623.0038730974168,0.009999136257463292

105,AVS57Xi,-512.9919764967757,0.00999840963347474

106,AVS57Yi,1229.004036019933,0.009999084802026513

107,AVS57Zi,1229.989063478223,0.009998540792827512

108,AVS58Xi,60.00591542750996,0.00999882019331969

109,AVS58Yi,1792.0026751331404,0.009999472846839545

110,AVS58Zi,1235.9924854883006,0.009998650982191343

111,AVS59Xi,629.9975096556891,0.009999411481711865

112,AVS59Yi,2360.997013573541,0.00999979769103441

113,AVS59Zi,1247.9997148928799,0.009999280598017624

114,AVS65Xi,-795.9941081630642,0.009998783143265131

115,AVS65Yi,945.0070449316535,0.009999186683921562

116,AVS65Zi,775.9890528345459,0.009998893654553687

117,AVS69Xi,357.99743878907054,0.009999097447533556

118,AVS69Yi,2092.996963763993,0.009999743773754897

119,AVS69Zi,783.9920618728692,0.009998915700155703

120,AVS70Xi,640.9973395401622,0.009999431330621648

121,AVS70Yi,2370.997766167578,0.009999859314952262

122,AVS70Zi,776.9969183194466,0.00999930724372695

123,AVS71Xi,955.9993903681435,0.009999498125767309

124,AVS71Yi,2681.999363175862,0.009999893252210092

125,AVS71Zi,796.9983078146444,0.009999406470394318

126,AVS75Xi,-793.9940018408455,0.009998832682146196

127,AVS75Yi,946.0046913097259,0.009999378304126495

128,AVS75Zi,401.99034816130734,0.009998943331262528

129,AVS91Xi,1318.0017500806543,0.00999967489124806

130,AVS91Yi,3040.000171076009,0.009999957606806697

131,AVS91Zi,-5.0006544505552055,0.00999964830748661

132,BVS10Xi,1940.9934647893533,0.009999259239719425

133,BVS10Yi,2766.010781954679,0.00999975452047458

134,BVS10Zi,1252.967849463882,0.00999922135304634

135,BVS16Xi,2746.99394066266,0.009997942950225427

136,BVS16Yi,1983.0051442863255,0.009999322973194405

137,BVS16Zi,878.9959044616988,0.009998680845437754

138,BVS35Xi,2239.9933257998878,0.009999332998929834

139,BVS35Yi,2480.000150915669,0.00999986042383599

140,BVS35Zi,-400.0038746445866,0.009999404664370874

141,BVS36Xi,1598.076452835577,0.9998680080650479

142,BVS36Yi,3113.6452739764823,2.3268757051343965

143,BVS36Zi,-359.7148769806604,0.946509849727181

144,BVS37Xi,1738.9965096344895,0.00999939832723616

145,BVS37Yi,2973.0001008211634,0.009999857886862343

146,BVS37Zi,-614.0072670590386,0.009999353101929996

147,BVS38Xi,2765.981755378779,0.009998170782131097

148,BVS38Yi,1967.9967079192475,0.009999560237218746

149,BVS38Zi,-810.0165295450636,0.009999041575945167

150,BVS42Xi,1604.9985524736921,0.009999449272878571

151,BVS42Yi,3101.001021402925,0.009999865473064733

152,BVS42Zi,-868.0044984295388,0.00999941307065333

153,BVS43Xi,2494.9890300737125,0.009998924147661304

154,BVS43Yi,2234.0000251602396,0.00999964356473114

155,BVS43Zi,-1205.0047318143609,0.009999278011223488

156,BVS44Xi,1966.9947804334222,0.009999512549416938

157,BVS44Yi,2747.0024452599255,0.009999836260761397

158,BVS44Zi,-1191.997409056887,0.009999543224801287

159,AVS104Xi,-760.9851755729808,0.009999062886828513

160,AVS104Yi,983.0116957494113,0.009999270571548409

161,AVS104Zi,-897.9943763319072,0.009999140095608235

162,AVS105Xi,-475.4333866146429,0.5973076637059802

163,AVS105Yi,1261.6945245267584,1.2157348495487883

164,AVS105Zi,-876.1208992979716,0.8471598341952145

165,AVS106Xi,-191.9982708081328,0.009998670786603605

166,AVS106Yi,1542.9991170715296,0.009999408908042408

167,AVS106Zi,-895.0056670573017,0.009998574306189978

168,AVS107Xi,100.99825458398954,0.00999931242645576

169,AVS107Yi,1833.998574695271,0.00999972073352705

170,AVS107Zi,-885.0056033625405,0.009999217548181397

171,AVS108Xi,391.10721170433413,0.7778228806010511

172,AVS108Yi,2122.8149919404505,1.6254523007706978

173,AVS108Zi,-912.1110880088877,0.8961776950297774

174,AVS116Xi,392.9882978221536,0.009999430451317508

175,AVS116Yi,2128.9955077359587,0.009999735987558994

176,AVS116Zi,-1352.0012546863063,0.009999371682493405

177,AVS117Xi,952.9937926320332,0.00999953444201709

178,AVS117Yi,2682.996924446994,0.009999848758447836

179,AVS117Zi,-1334.0055585488697,0.009999505986941378

## Github Repository

## <https://github.com/jnaess/ENGO531.git>

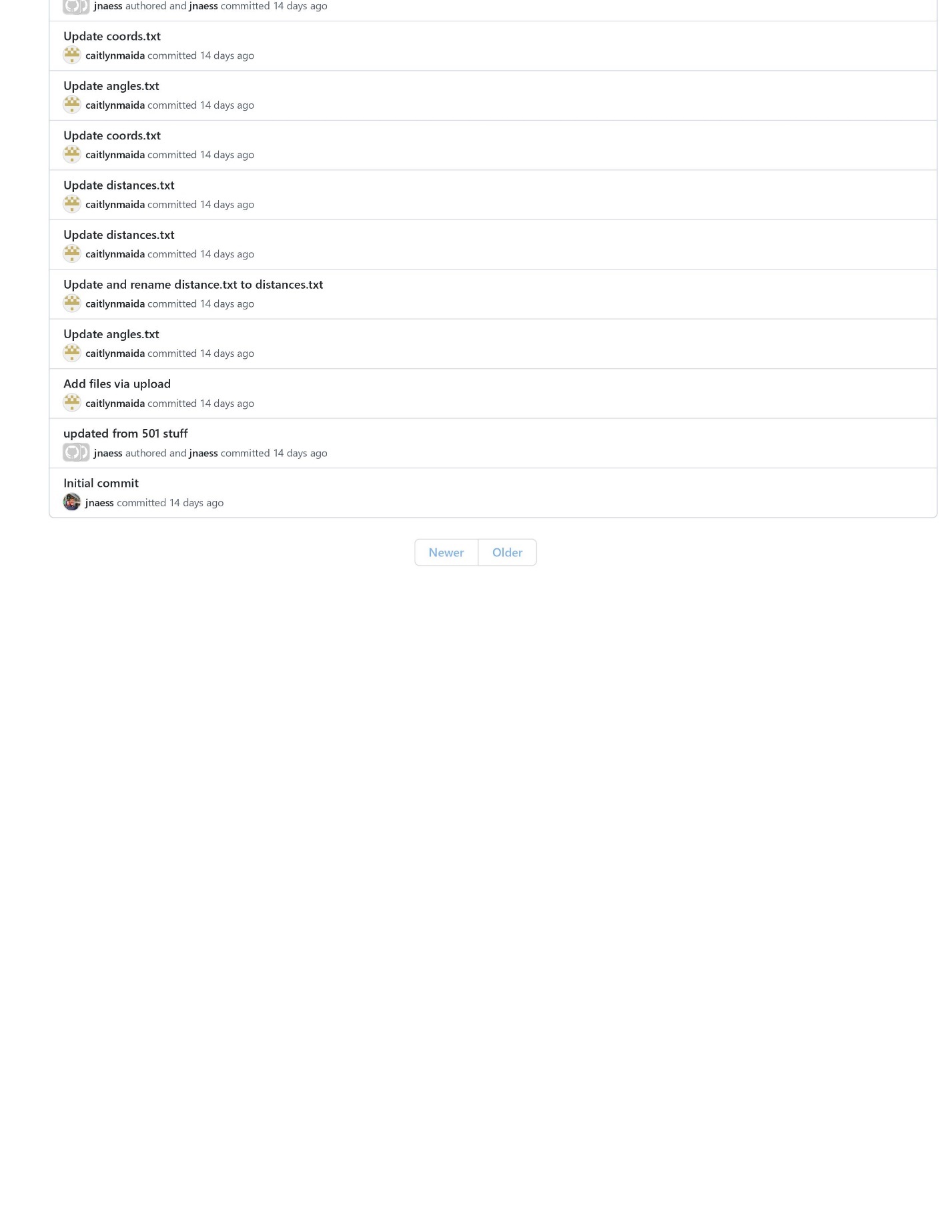
## Github Commits

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated



## Tools.py

import numpy as np

import matplotlib.pyplot as plt

import os

class Tools():

"""

Desc:

This class was made as a toolbox for plotting and converting values

"""

def \_\_init\_\_(self):

"""

Just exsists :-)

"""

def plot\_mat(self, matrix, title = "Title", round\_to = 6):

"""

Desc:

Checks to see if a "Figures" folder has been made. If it is not made then it makes it.

Then saves the input matrix as a .png to the folder with "Title" as the name

Input:

matrix: the numpy array to plot

title: the title of the array and output image (default "Title")

round\_to: decimals to round to (default 6)

Output:

"""

#set up figure with decently sized boxes

fig, ax = plt.subplots(figsize = (10,15))

ax.imshow(matrix)

plt.title(title)

# Loop over data dimensions and create text annotations.

for i in range(matrix.shape[0]):

for j in range(matrix.shape[1]):

#inputs numerical values

text = ax.text(j, i, round(matrix[i, j],round\_to),

ha="center", va="center", color="w")

#plt.axis('off')

#folder is just called figures

folder\_path = 'Figures/'

file\_name = title

#makes folder if not already there

if not os.path.isdir(folder\_path):

os.makedirs(folder\_path)

#saves to the folder using the title name

fig.savefig(os.path.join(folder\_path,file\_name))

plt.figure().clear()

plt.close()

plt.cla()

plt.clf()

def save\_df(self, df, title):

"""

Desc:

saves df to the file

Input:

Output:

"""

#folder is just called files

folder\_path = 'Files/'

file\_name = title

#makes folder if not already there

if not os.path.isdir(folder\_path):

os.makedirs(folder\_path)

#saves to the folder using the title name

df.to\_csv(os.path.join(folder\_path,file\_name))Tables.py

from scipy import misc

from scipy import stats

import pandas as pd

import numpy as np

class Tables():

"""

Parent class to PostAdjustmentTester which generates the significant values to increase modularity

"""

def \_\_init\_\_(self):

"""

"""

def newtons\_method(self, x, tolerance=0.0001):

while True:

x1 = x - self.f(x) / misc.derivative(self.f, x)

t = abs(x1 - x)

if t < tolerance:

break

x = x1

return x

def f(self, x):

return 1 - stats.chi2.cdf(x, self.r) - self.pvalue

def x\_2(self):

"""

Reference:

Code reformatted to return a single line of the desired x\_2 value based on our DOF (instead of a given value)

Code refers to functions "newtons\_method", "f", "x\_2"

https://moonbooks.org/Articles/How-to-create-a-Chi-square-table-using-python-/

Desc:

returns a chi-square dataframe row for the designated DOF

Input:

r: defrees of freedom

Output:

"""

self.pvalueList = [0.995, 0.99, 0.975, 0.95, 0.90, 0.10, 0.05, 0.025, 0.01, 0.005]

results = []

for i in range(self.r,self.r+1):

self.r = i

Result = []

for self.pvalue in self.pvalueList:

x0 = self.r # x0 approximation

x = self.newtons\_method(x0)

Result.append(x)

for i in range(10):

Result[i] = round(Result[i],3)

results.append(Result)

return pd.DataFrame(results, columns = self.pvalueList)

## Net.py

from numpy import transpose as t

from numpy import matrix as mat, matmul as mm

from numpy import linalg as lin

from numpy.linalg import inv

import math as m

import numpy as np

import pandas as pd

from LeastSquares import LS

from Level import Delta

from PostAdjustmentTester import PostAdjustmentTester

class Network(LS, PostAdjustmentTester):

"""

Build to run the least squares adjustment and set up the overall network

"""

def \_\_init\_\_(self, models, net\_type = "Photo"):

"""

Desc:

Input:

models: list of models that have been initialized with

data. Must contain the same number of columns in their a

matrix (predefined by LS())

Output:

"""

LS.\_\_init\_\_(self)

PostAdjustmentTester.\_\_init\_\_(self)

#for picking things

self.net\_type = net\_type

self.models = models

if self.net\_type == "Photo":

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_setup first round of stuff\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

self.initialize\_variables()

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_begin LSA\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

self.photo\_LSA()

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_format matrices for outputting statistics\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

self.photo\_mats()

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_output statistics\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

self.final\_matrices()

def initialize\_variables(self):

"""

Desc:

initializes major variables (combining matrices and stuff)

Input:

Output:

self.u

"""

self.ue = self.models[0].ue

self.uo = self.models[0].uo

self.u = self.models[0].ue + self.models[0].uo

#set up observation matrix

temp = []

for obs in self.models:

temp.append(obs.obs)

self.obs = np.vstack(temp)

#set up errors matrix

temp = []

for obs in self.models:

temp.append(obs.errs)

self.errs = np.vstack(temp)

if self.net\_type == "Photo":

#set up control weight errors matrix

temp = []

for obs in self.models:

temp.append(obs.errs\_o)

self.errs\_o = np.vstack(temp)

#set up number of observations variable

self.n = len(self.errs)

#set up design matrix

self.design()

#set up covariance (no additional formatting needed)

self.covariance()

#set up apriori

self.apriori = 1

#set up weight matrix

self.P = self.apriori\*\*2 \* inv(self.Cl)

if self.net\_type == "Photo":

#then a Po will also need to be made

self.Po = mat(np.zeros((self.uo, self.uo)))

for i in range(0,self.uo):

if self.errs\_o[i] != 0:

self.Po[i,i] = 1/self.errs\_o[i]\*\*2

def final\_matrices(self):

"""

Desc:

Once the LSA is completed then this generates all desired matrices for analysis

Input:

Output:

self.r\_hat: residuals

self.l\_hat: adjusted observations

self.a\_post: a-posteriori variance factor

self.uvf: unit variance factor

self.Cx (also Cs):

self.Cl:

self.Cr:

"""

self.r\_hat = mm(self.A,self.S\_hat) + self.w\_0

self.l\_hat = self.obs + self.r\_hat

self.a\_post = m.sqrt(mm(t(self.r\_hat),mm(self.P,self.r\_hat)/(self.n-self.u))[0,0])

self.uvf = self.a\_post\*\*2 / self.apriori\*\*2

self.Cx = self.a\_post\*\*2 \* inv(mm(t(self.A),mm(self.P,self.A)))

if self.net\_type == "Photo":

self.Cx = inv(self.N)

#self.plot\_mat(self.Cx, "Covariance Matrix of Unknowns")

self.Cl = mm(self.A,mm(self.Cx,t(self.A)))

#self.plot\_mat(self.Cl, "Covariance Matrix of Measurements")

self.Cr = self.a\_post\*\*2\*inv(self.P)-self.Cl

#self.plot\_mat(self.Cr, "Covariance Matrix of Residuals")

def nonlinear\_LSA(self):

"""

Desc:

Iterates a nonlinear LSA, checking whether criterea was met. Once it was met then it constructs the final matrices for analysis

Input:

Output:

"""

self.not\_met = True

i = 0

self.w\_0 = mat(np.zeros((self.n, 1)))

self.S\_hat = mat(np.zeros((self.n, 1)))

self.x\_hat = mat(np.zeros((self.n, 1)))

while self.not\_met:

i = i + 1

#print("LSA iteration: " + str(i))

#print("x\_0: ")

#print(LS.x\_0)

#l\_0

self.obs\_0()

#update l\_0 and A

self.update\_values()

#misclosure

self.w\_0 = self.l\_0 - self.obs

#S\_hat

self.S\_hat = -mm(inv(mm(t(self.A),mm(self.P,self.A))),mm(t(self.A),mm(self.P,self.w\_0)))

#print("l\_0: ")

#print(self.l\_0)

#x\_hat

self.x\_hat = LS.x\_0 + self.S\_hat

#update x\_0

LS.x\_0 = self.x\_hat

#print("S\_hat:")

#print(self.S\_hat)

#print("x\_hat: ")

#print(self.x\_hat)

#print("A: ")

#print(self.A)

self.convergence(i)

#print("LSA passed in: " + str(i) + " iterations")

#self.final\_matrices()

def photo\_LSA(self):

"""

Desc:

Iterates a nonlinear LSA, checking whether criterea was met. Once it was met then it constructs the final matrices for analysis

Input:

Output:

"""

self.not\_met = True

i = 0

self.w\_0 = mat(np.zeros((self.n, 1)))

self.S\_hat = mat(np.zeros((self.n, 1)))

self.x\_hat = mat(np.zeros((self.n, 1)))

while self.not\_met and i < 5:

i = i + 1

#l\_0

self.obs\_0()

#update l\_0 and A

self.update\_values()

#misclosure

self.w\_0 = self.l\_0 - self.obs

self.set\_N()

self.set\_U()

#S\_hat

self.S\_hat = -mm(inv(self.N),self.U)

#x\_hat

self.x\_hat = LS.x\_0 + self.S\_hat

#update x\_0

LS.x\_0 = self.x\_hat

self.convergence(i)

print("LSA passed in: " + str(i) + " iterations")

#self.final\_matrices()

#not 100% sure but probably

self.A = self.N

def error\_ellipses(self):

"""

Desc:

generates the error ellipses, minor, major, bearing\_major

\*\*must already have self.Cx generates\*\*

\*\*assumes Xa, Ya, Xb, Yb, Xc, Yc etc in the Cx diagonal\*\*

Input:

Output:

"""

self.u

ellipses = []

for i in range(0,self.Cx.shape[0],2):

q11 = self.Cx[i,i]

q12 = self.Cx[i,i+1]

q21 = self.Cx[i+1,i]

q22 = self.Cx[i+1,i+1]

ellipses.append(self.ellipse(q11, q12, q21, q22))

return ellipses

def ellipse(self, q11, q12, q21, q22):

"""

Desc:

Calculates the error ellipse, returns back a dataframe of the values

Input:

q11,

q12,

q21,

q22

Output:

{

"minor": float,

"major": float,

"major\_orientation": radians

}

"""

minor = m.sqrt(abs((q11 + q22 - m.sqrt((q11-q22)\*\*2+4\*(q12\*\*2)))/2))

major = m.sqrt(abs((q11 + q22 + m.sqrt((q11-q22)\*\*2+4\*(q12\*\*2)))/2))

major\_orientation = m.atan(q12/(major\*\*2-q22))

return {

"minor": minor,

"major": major,

"major\_orientation": major\_orientation

}

def convergence(self,i):

"""

Desc:

Checks based on this criterea, if convergence is met then sets self.not\_met to False

Input:

i: number of iterations (for simple # of ter break)

Output:

self.not\_met --> False if the criterea is met

"""

#max 10 iterations

if i > 3:

self.not\_met = False

#minimum self.S\_hat to be under .001m

not\_under = False

for key in self.S\_hat:

if abs(key[0,0]) > .0001:

#this means the criterea was not met for atleast one of the unknowns

not\_under = True

if not not\_under:

#then all things were under .0001m in change and therefore the criterea was met

self.not\_met = False

def covariance(self):

"""

Desc:

Initialized covariance matrix based on observation standard deviations

Input:

Output:

self.Cl

"""

self.Cl = mat(np.zeros((self.n, self.n)))

for i in range(0,self.n):

self.Cl[i,i] = self.errs[i]\*\*2

def photo\_mats(self):

"""

Desc:

Sets up matrices needed for statistics

Input:

Output:

self.A

self.S

"""

self.A = np.concatenate((self.Ae,self.Ao), axis = 1)

self.u\_list = []

for i in self.models[0].u\_list\_ae:

self.u\_list.append("{}Xcj".format(i))

self.u\_list.append("{}Ycj".format(i))

self.u\_list.append("{}Zcj".format(i))

self.u\_list.append("{}w".format(i))

self.u\_list.append("{}o".format(i))

self.u\_list.append("{}k".format(i))

for i in self.models[0].u\_list\_ao:

self.u\_list.append("{}Xi".format(i))

self.u\_list.append("{}Yi".format(i))

self.u\_list.append("{}Zi".format(i))

def design(self):

"""

Desc:

Set up overall design matrix

Input:

Output:

self.A

"""

#self.A = mat(np.zeros((self.n, self.u)))

#temp = []

#for model in self.models:

# temp.append(model.A)

#self.A = np.vstack(temp)

self.Ae = self.models[0].Ae

self.Ao = self.models[0].Ao

def set\_N(self):

"""

Desc:

Sets up the N matrix with the four quadrants

Input:

self.Ae

self.Ao

self.P

Output:

self.Nee

self.Neo

self.Noo

self.N

"""

self.Nee = mm(t(self.Ae),mm(self.P,self.Ae))

self.Neo = mm(t(self.Ae),mm(self.P,self.Ao))

self.Noo = mm(t(self.Ao),mm(self.P,self.Ao))+self.Po

a = np.concatenate((self.Nee,self.Neo), axis = 1)

b = np.concatenate((t(self.Neo),self.Noo), axis = 1)

self.N = np.concatenate((a,b), axis = 0)

def set\_U(self):

"""

Desc:

Sets up the U matrix with the two halves

Input:

self.Ae

self.Ao

self.P

self.w\_0

Output:

self.Ue

self.Uo

self.U

"""

#self

self.w\_0\_o = LS.x\_0\_ao - self.x\_0[self.ue:,0]

self.Ue = mm(t(self.Ae),mm(self.P,self.w\_0))

self.Uo = mm(t(self.Ao),mm(self.P,self.w\_0))+mm(self.Po,self.w\_0\_o)

#print(self.Uo.shape)

self.U = np.concatenate((self.Ue,self.Uo), axis = 0)

def n\_mat(self):

"""

"""

self.N = mm(t(self.A),mm(self.P,self.A))

def cx\_mat(self):

"""

"""

self.Cx = inv(self.N)

def w\_mat(self):

"""

"""

#adds constants and unknowns together and solves for values

self.w = mm(self.A,LS.x\_0) - self.obs

#\_\_\_\_\_\_\_\_\_\_\_\_for non linear this will need to change\_\_\_\_\_\_

def u\_mat(self):

"""

"""

self.v = t(self.A,mm(self.P,self.w))

def correction(self):

"""

"""

self.S = -mm(inv(self.N),mm(t(self.A),mm(self.P,self.w)))

def obs\_0(self):

"""

Desc:

Assembles l\_obs from each matrix

Input:

Output:

self.l\_0 constructed

"""

self.l\_0 = mat(np.zeros((self.n, 1)))

temp = []

for obs in self.models:

temp.append(obs.l\_0)

self.l\_0 = np.vstack(temp)

def update\_values(self):

"""

Desc:

Updates x\_0 and design and l\_0

Input:

Uses most recent x\_hat value

Output:

none:

"""

#update models

for model in self.models:

#model.x\_0 = self.x\_0

model.obs\_0()

#update design matrix

model.set\_design()

#update within network

self.design()

self.obs\_0()

## LeastSquares.py

from numpy import transpose as t

from numpy import matrix as mat, matmul as mm

import math as m

import numpy as np

import pandas as pd

from Tools import Tools

class LS(Tools):

"""

Holds the universal values needed to integrate the different LS adjustments into one

"""

x\_0 = []

def \_\_init\_\_(self, file\_name = "coords.txt", debugging = False):

"""

Desc:

reads in the list of knowns and unknowns and assigns their values. Will construct design matrix, etc. based off of these

Input:

file\_name where the knowns and unknowns are defined

debugging, T/F. If true then more printing of stuff happens

Output:

sets up u\_list (predefined in here)

sets up number of unknowns (self.u)

"""

#brings in the tool files for use

Tools.\_\_init\_\_(self)

self.debugging = debugging

self.file\_name = file\_name

#self.read\_2D()

def read\_2D(self):

"""

Desc:

reads in the 2D set of points and assigns values

expects format of [name easting northing known/unknown]

more specifically: [Point X[m] Y[m] Known[n]/Unknown[u]]

Input:

self.file\_name

Output:

self.u\_list (string list of unknown)

self.x\_0 (initial guesses of unknowns)

self.c (constant values of knowns)

self.datums (string list of knowns)

self.u # of unknowns

"""

df = pd.read\_csv(self.file\_name, sep = ' ')

#currently only formatted for 2D

self.u\_list = []

LS.x\_0 = []

self.c = []

#pretty sure datums aren't actually used

self.datums = []

#assign values

for index, row in df.iterrows():

#check if known or unknown

if row[3] == "u":

#unknown name

self.u\_list.append(row[0]+"\_E")

self.u\_list.append(row[0]+"\_N")

#add unknown values in order of x, y

LS.x\_0.append(row[1])

LS.x\_0.append(row[2])

else: #then they are "n" --> knowns

#known name

self.datums.append(row[0]+"\_E")

self.datums.append(row[0]+"\_N")

#add known values in order of x, y

self.c.append(row[1])

self.c.append(row[2])

LS.x\_0 = t(mat(LS.x\_0))

self.c = t(mat(self.c))

self.u = len(self.u\_list)

def find\_col(self, dimension, point\_name, li = "u"):

"""

Desc:

returns the column index of the desired points

expects 'n' for known and 'u' for unknown

\*\*all values must be in caps\*\*

Input:

u\_list, list of strings of "pointname\_dimension"

dimension, string either "N", "E", "H"

Output:

integer value of the column to place the value

in the desired design matrix

"""

if li == "u" :

li = self.u\_list

else:

li = self.datums

index = 0

for key in li:

#split the key into point name and dimension

temp\_name = key.split('\_')[0]

temp\_dimension = key.split('\_')[1]

if (point\_name == temp\_name and dimension == temp\_dimension):

return index

else:

index = index + 1

#debugging stuff

if self.debugging:

print(point\_name + " Could not be found")

return -1

def set\_col\_list\_ae(self):

"""

Desc:

Initializes the order of image\_id's for the Ae matrix so that numbers are positioned correctly

Input:

Output:

self.u\_list\_ae for Ae

"""

#assumes images already sorted in ascending order

self.u\_list\_ae = self.pho['image\_id'].unique()

def find\_col\_ae(self, image\_id, li = "u"):

"""

Desc:

returns the column index of the desired points

expects 'n' for known and 'u' for unknown

\*\*all values must be in caps\*\*

Input:

u\_list\_ae, list of strings of "pointname\_dimension"

image\_id: string of the image id index to return

Output:

integer value of the column to place the value in the desired design matrix multiplied by 6

"""

if li == "u" :

li = self.u\_list\_ae

else:

li = self.datums

index = 0

for key in li:

if image\_id == key:

return index\*6

else:

index = index + 1

def set\_col\_list\_ao(self):

"""

Desc:

Initializes the order of point\_id's for the Ao matrix so that numbers are positioned correctly from all points observed (unique values for columns)

Input:

Output:

self.u\_list\_ao for Ao

"""

#assumes images already sorted in ascending order

self.u\_list\_ao = self.obj['point\_id'].unique()

def find\_col\_ao(self, point\_id, li = "u"):

"""

Desc:

returns the column index of the desired points

expects 'n' for known and 'u' for unknown

\*\*all values must be in caps\*\*

Input:

u\_list\_ao, list of strings of "pointname\_dimension"

point\_id: string of the image id index to return

Output:

integer value of the column to place the value in the desired design matrix multiplied by 3 for XYZ

"""

if li == "u" :

li = self.u\_list\_ao

else:

li = self.datums

index = 0

for key in li:

if point\_id == key:

return index\*3

else:

index = index + 1

## PostAdjustmentTester.py

from numpy import transpose as t

from numpy import matrix as mat, matmul as mm

import matplotlib as plt

from numpy import linalg as lin

from numpy.linalg import inv

import math as m

import numpy as np

import pandas as pd

from LeastSquares import LS

from Level import Delta

from Tables import Tables

from scipy import stats as st

from scipy.stats import t as stu

from scipy.stats import chi2

class PostAdjustmentTester(Tables):

"""

Desc:

Assumes that the LSA has been conducted and outputs results for post adjustment tests

"""

def \_\_init\_\_(self):

"""

Desc:

Figuring out if we need to take in matrices or if we'll just inherit the class and assume that they're build

"""

Tables.\_\_init\_\_(self)

def global\_a\_posteriori(self, alpha = .05):

"""

Desc:

Tests the statistical sifnigicance of the aposteriori to a priori variance factor

Input:

alpha: to generate the two confidence intervals. Be sure to make sure that these values are generated in the respective dataframe of values, otherwise they won't be found :-)

self.u: # of unknowns

self.n: # of observations

self.a\_post: final computed a posteriori variance factor

self.apriori: initial apriori variance factor

Output:

Prints the output and respective indication

"""

#set up DOF (r)

self.r = self.n - self.u

#retrieves dataframe of chi values for our respective DOF

ch\_df = self.x\_2()

low = ch\_df[alpha][0]

high = ch\_df[1-alpha][0]

y = (self.r \* self.a\_post\*\*2)/self.apriori\*\*2

#if fails this check then there is an indication that the residuals or math model may be off

print("{} tested with chi\_square boundries of {} and {}".format(y, low, high))

if y > low and y < high:

print("Global A-Posteriori Variance Factor Test passes at a {} confidence level".format((1 - alpha)\*100))

print("There is no indication for errors within residual or the math models")

else:

print("Global A-Posteriori Variance Factor Test \*\*failed\*\* at a {}% confidence level".format((1 - alpha)\*100))

print("There is indication that errors exsist within residual or the math models")

def significance\_estimated\_param(self, alpha = .05):

"""

Desc:

Determines whether there is statistical signifiance to beleive the final estimated value of parameters

Input:

alpha: to generate the two confidence intervals. Be sure to make sure that these values are generated in the respective dataframe of values, otherwise they won't be found :-)

self.n: # of observations

self.x\_hat

self.u\_list: for labelling

self.Cx: for extracting std dev values of parameters

Output:

retrunds dataframe of values [Unknown Final Value Value Standard Deviation Test Value Indicated Significance Alpha Tested Confidence Level Test Bounds]

"""

#set up DOF (r)

self.r = self.n - self.u

high = stu.ppf(1.0 - alpha, self.r)

low = stu.ppf(alpha, self.r)

#final paramter values

xs = []

#unknown names in string format

us = []

#list to store their signifiance as Signifiance or Not Significant

sig = []

#list to store the value that was checked

sig\_value = []

#list of standard deviation values

std = []

#test values

y = []

#confidence levels

conf = []

#confidence levels

alphs = []

#test bounds

bounds = []

for i in range(0,self.u):

std.append(m.sqrt(self.Cx[i,i]))

y.append((self.x\_hat[i]/std[i])[0,0])

if y[i] > low and y[i] < high:

#if fails then there IS statistical significance

sig.append("No")

else:

sig.append("Yes")

xs.append(self.x\_hat[i][0,0])

us.append(self.u\_list[i])

conf.append((1-alpha)\*100)

alphs.append(alpha)

bounds.append(str([low, high]))

#to store values in a dictionary before conversion to dataframe

dict\_list = {

"Unknown": us,

"Final Value": xs,

"Value Standard Deviation": std,

"Test Value": y,

"Indicated Significance": sig,

"Alpha Tested": alphs,

"Confidence Level": conf,

"Test Bounds": bounds

}

#return dict\_list

return pd.DataFrame.from\_dict(dict\_list)

def semi\_global\_residuals(self, alpha = .05):

"""

Desc:

Conducts the semi global test on residuals, also known as the gooness-of-fit or normality test on residuals

Input:

self.r\_hat: residuals

alpha = .05: to find confidence level

self.Cr: extracting std of residuals

self.n: number of observations

Output:

prints whether the test passed and the reccomended interpretation

"""

#normalize residuals

norm\_r = []

for i in range(0,self.n):

norm\_r.append(self.r\_hat[i,0]/m.sqrt(self.Cr[i,i]))

#number of bins

M = round(m.sqrt(self.n))

counts, bins = np.histogram(norm\_r, bins = M)

#compute estimated number of residuals per bin

e = []

for i in range(M):

#get probability of total bin

p\_start = st.norm.cdf(bins[i])

p\_end = st.norm.cdf(bins[i+1])

p = p\_end - p\_start

#append total number of expected residuals

e.append(p\*self.n)

#compute X\_2 for each bin

chis = []

for i in range(M):

chis.append((e[i]-counts[i])\*\*2/e[i])

#sum all chis for test statistic y

y = sum(chis)

#conduct statistical test

dof = M - 1

prob = 1 - alpha

chi = chi2.ppf(prob, dof)

print("{} tested with chi\_square of {} ".format(y, chi))

if y > chi:

print("The Semi-Global, goodness-of-fit test on the residuals \*\*Failled\*\*")

print("There is a sign that either there are outliers or the functional model was not appropriate for the data set")

else:

print("The Semi-Global, goodness-of-fit test on the residuals \*\*Passed\*\*")

print("There is no sign of outliers or functional model errors")

#plt.hist(norm\_r, m)

def blunder\_detection(self, alpha = .01):

"""

Desc:

Conducts the local test on the residuals, aka blunder detection

Input:

alpha = .01: for 99% confidence of a blunder

self.Cr: for extracting std of residuals

self.r\_hat: for extracting residuals

Output:

Returns a dataframe with columns ["Observation", "Outlier", "Test Value", "Test Bounds"]

"""

#statistical test values

low = st.norm.ppf(alpha/2)

high = st.norm.ppf(1-alpha/2)

#normalize residuals (test statistic)

y = []

#list of Yes or No outliers

outlier = []

#confidence levels

conf = []

#observations

observations = []

#test bounds

bounds = []

for i in range(0,self.n):

#for DF

observations.append(i)

bounds.append(str([low, high]))

conf.append((1-alpha)\*100)

y.append(self.r\_hat[i,0]/m.sqrt(self.Cr[i,i]))

if y[i] > low and y[i] < high:

#passes test --> not an outlier

outlier.append("No")

else:

outlier.append("Yes")

dic = {

"Observation": observations,

"Outlier": outlier,

"Confidence Level": conf,

"Test Value": y,

"Test Bounds": bounds

}

return pd.DataFrame.from\_dict(dic)

def final\_file(self, alpha = .05):

"""

Desc:

Final Dataframe File

Input:

alpha: to generate the two confidence intervals. Be sure to make sure that these values are generated in the respective dataframe of values, otherwise they won't be found :-)

self.n: # of observations

self.x\_hat

self.u\_list: for labelling

self.Cx: for extracting std dev values of parameters

Output:

retrunds dataframe of values [Unknown Final Value Value Standard Deviation Test Value Indicated Significance Alpha Tested Confidence Level Test Bounds]

"""

#set up DOF (r)

self.r = self.n - self.u

high = stu.ppf(1.0 - alpha, self.r)

low = stu.ppf(alpha, self.r)

#final paramter values

xs = []

#unknown names in string format

us = []

#list to store their signifiance as Signifiance or Not Significant

sig = []

#list to store the value that was checked

sig\_value = []

#list of standard deviation values

std = []

#test values

y = []

#confidence levels

conf = []

#confidence levels

alphs = []

#test bounds

bounds = []

for i in range(0,self.u):

std.append(m.sqrt(self.Cx[i,i]))

y.append((self.x\_hat[i]/std[i])[0,0])

if y[i] > low and y[i] < high:

#if fails then there IS statistical significance

sig.append("No")

else:

sig.append("Yes")

xs.append(self.x\_hat[i][0,0])

us.append(self.u\_list[i])

conf.append((1-alpha)\*100)

alphs.append(alpha)

bounds.append(str([low, high]))

#to store values in a dictionary before conversion to dataframe

dict\_list = {

"Unknown": us,

"Final Value (mm or rad)": xs,

"Value Standard Deviation (mm or rad)": std,

#"Test Value": y,

#"Indicated Significance": sig,

#"Alpha Tested": alphs,

#"Confidence Level": conf,

#"Test Bounds": bounds

}

#return dict\_list

return pd.DataFrame.from\_dict(dict\_list)

## Design\_o.py

from numpy import matrix as mat, matmul as mm

from numpy import transpose as t

import math as m

import numpy as np

import pandas as pd

from Bundle import Bundle

from Design\_e import Design\_e as ae

from LeastSquares import LS

class Design\_o(Bundle, LS):

"""

Desc:

Generates and facilitates the manipulation of Ae

"""

def \_\_init\_\_(self):

"""

Desc:

Input:

Output:

"""

Bundle.\_\_init\_\_(self)

LS.\_\_init\_\_(self)

self.initialial\_setup()

def initialial\_setup(self):

"""

Desc:

initializes major variables (combining matrices and stuff)

Input:

Output:

self.u

"""

self.xp = self.pix\_to\_m\*self.int["xp"][0]

self.yp = self.pix\_to\_m\*self.int["yp"][0]

self.c = self.pix\_to\_m\*self.int["c"][0]

#from LS class to find unknown columns

self.set\_col\_list\_ao()

self.set\_col\_list\_ae()

self.set\_X\_0()

self.set\_obs()

self.obs\_0()

self.set\_design()

def set\_obs(self):

"""

Desc:

uses self.pho to take the x and y and set up the observations and converts them to RHC with a bundle functions

sets control point to .01mm and current tie points to 10mm

Input:

self.pho

Output:

self.obs: l matrix (never changes)

self.errs

"""

self.obs = mat(np.zeros((self.n, 1)))

#data input as \*\*\*mm\*\*\*

self.errs = mat(np.zeros((self.n, 1)))

#get desired numbers in a list

y = self.pho['y'].to\_list()

x = self.pho['x'].to\_list()

check = self.pho['knowns'].to\_list()

j = 0

for i in range(0, self.n, 2):

#set up x\_ij and y\_ij info

self.rhc(x[j],y[j])

#if j == 0:

#print("xp: {} | yp: {} | xmm: {} | ymm: {}".format(x[j], y[j], self.x\_ij, self.y\_ij))

#x pixel

self.obs[i,0] = self.x\_ij

#y pixel

self.obs[i+1,0] = self.y\_ij

self.errs[i,0] = .00345

self.errs[i+1,0] = .00345

#assign errors

j = j+1

self.set\_control\_weights()

def set\_control\_weights(self):

"""

Desc:

Sets control weights for datum definition

Input:

Output:

self.errs\_o

"""

#for Po

self.errs\_o = mat(np.zeros((self.uo, 1)))

#to skip the Ae ones (only pixel points wanted)

check = self.pho['knowns'].to\_list()

j = self.ue

for i in range(0,self.uo,3):

# print(str(i)+" "+str(self.ue)+" "+str(self.uo))

if check[j] == "u":

#then tie point and larger std

self.errs\_o[i,0] = 0

self.errs\_o[i+1,0] = 0

self.errs\_o[i+2,0] = 0

else:

#control points given extra weight

self.errs\_o[i,0] = .01

self.errs\_o[i+1,0] = .01

self.errs\_o[i+2,0] = .01

#increment index in y and x lsits

j = j+1

def set\_X\_0(self):

"""

Desc:

Sets up X\_0 from the dataframe values

Input:

Output:

self.x\_0

and

LS.x\_0

"""

#assumes images already sorted in ascending order

#assumes camera also sorted

x\_0\_ae = []

for index, row in self.ext.iterrows():

x\_0\_ae.append(row["Xc"])

x\_0\_ae.append(row["Yc"])

x\_0\_ae.append(row["Zc"])

x\_0\_ae.append(m.radians(row["w"]))

x\_0\_ae.append(m.radians(row["o"]))

x\_0\_ae.append(m.radians(row["k"]))

x\_0\_ao = []

for index, row in self.obj.iterrows():

x\_0\_ao.append(row["X"])

x\_0\_ao.append(row["Y"])

x\_0\_ao.append(row["Z"])

LS.x\_0\_ao = t(mat(x\_0\_ao))

self.x\_0 = t(mat(x\_0\_ae+x\_0\_ao))

LS.x\_0 = self.x\_0

def obs\_0(self):

"""

desc:

Sets up self.l\_0 (extimated observations)

Used for finding the current misclosure

Assumes only one camera for IOP's from self.int

input:

self.x\_0

output:

self.l\_0

"""

self.rhc(self.int["xp"][0], self.int["yp"][0])

self.xp = self.x\_ij

self.yp = self.y\_ij

self.c = self.pix\_to\_m\*self.int["c"][0]

#set it up as just zeros

self.l\_0 = mat(np.zeros((self.n, 1)))

for i in range(0, self.n, 2):

obs = self.pho.iloc[int(i/2)]

#row for ae parameters

j = self.find\_col\_ae(obs["image\_id"])

#row for ue parameters

j\_2 = self.ue + self.find\_col\_ao(obs["point\_id"])

self.X\_cj = LS.x\_0[j]

self.Y\_cj = LS.x\_0[j+1]

self.Z\_cj = LS.x\_0[j+2]

self.w = LS.x\_0[j+3]

self.o = LS.x\_0[j+4]

self.k = LS.x\_0[j+5]

#xp, yp, c values should be updated here if multiple cameras were used

self.X\_i = LS.x\_0[j\_2]

self.Y\_i = LS.x\_0[j\_2+1]

self.Z\_i = LS.x\_0[j\_2+2]

#if i == 0:

#print("xp: {} | yp: {} | c: {} | X\_cj: {} | Y\_cj: {} | Z\_cj: {} | w: {} | o: {} | k: {} | X\_i: {} | Y\_i: {} | Z\_i: {}".format(self.xp, self.yp, self.c, self.X\_cj, self.Y\_cj, self.Z\_cj, self.w, self.o, self.k, self.X\_i,self.Y\_i,self.Z\_i))

v = self.V()

w = self.W()

u = self.U()

m\_temp = self.M()

#if i == 0:

#print("xp: {} | yp: {} | c: {} | u: {} | w: {} | v: {}".format(self.xp, self.yp, self.c, u,w,v))

x = self.xp - self.c\*u/w

y = self.yp - self.c\*v/w

#setup xij

self.l\_0[i,0] = x

#set up yij

self.l\_0[i+1,0] = y

def set\_design(self):

"""

Desc:

Initializes the design matrix

Output:

Input:

"""

self.xp = self.pix\_to\_m\*self.int["xp"][0]

self.yp = self.pix\_to\_m\*self.int["yp"][0]

self.c = self.pix\_to\_m\*self.int["c"][0]

self.update\_Ae()

#set it up as just zeros

self.Ao = mat(np.zeros((self.n, self.uo)))

#0, 2, 4, etc. are X pixels

#1, 3, 5, etc. are Y pixels

#\_\_print("n: "+str(self.n))

for i in range(0, self.n, 2):

#increments every two because one row is for X, one row is for Y

#each time we should go through one observation

#indexes every 2

#this is the observation

#get image id from photo obs

obs = self.pho.iloc[int(i/2)]

#get image row from ext EOP's

#j = int(obs["image\_id"])

j = self.find\_col\_ao(obs["point\_id"])

j\_2 = self.find\_col\_ae(obs["image\_id"])

#then evens (X partial)

self.Ao[i,j] = -self.Ae[i,j\_2]

#Y

self.Ao[i,j + 1] = -self.Ae[i,j\_2+1]

#Z

self.Ao[i,j + 2] = -self.Ae[i,j\_2+2]

#then odds (Y partial)

#X

self.Ao[i+1,j] = -self.Ae[i+1,j\_2]

#Y

self.Ao[i+1,j + 1] = -self.Ae[i+1,j\_2+1]

#Z

self.Ao[i+1,j + 2] = -self.Ae[i+1,j\_2+2]

def update\_Ae(self):

"""

Desc:

Initializes the design matrix

Input:

LS.x\_0

Output:

"""

self.xp = self.pix\_to\_m\*self.int["xp"][0]

self.yp = self.pix\_to\_m\*self.int["yp"][0]

self.c = self.pix\_to\_m\*self.int["c"][0]

#set it up as just zeros

self.Ae = mat(np.zeros((self.n, self.ue)))

#0, 2, 4, etc. are X pixels

#1, 3, 5, etc. are Y pixels

#\_\_print("n: "+str(self.n))

for i in range(0, self.n, 2):

obs = self.pho.iloc[int(i/2)]

#row for ae parameters

j = self.find\_col\_ae(obs["image\_id"])

#row for ue parameters

j\_2 = self.ue + self.find\_col\_ao(obs["point\_id"])

self.X\_cj = LS.x\_0[j]

self.Y\_cj = LS.x\_0[j+1]

self.Z\_cj = LS.x\_0[j+2]

self.w = LS.x\_0[j+3]

self.o = LS.x\_0[j+4]

self.k = LS.x\_0[j+5]

#xp, yp, c values should be updated here if multiple cameras were used

self.X\_i = LS.x\_0[j\_2]

self.Y\_i = LS.x\_0[j\_2+1]

self.Z\_i = LS.x\_0[j\_2+2]

v = self.V()

w = self.W()

u = self.U()

m\_temp = self.M()

#then evens (X partial)

#X

self.Ae[i,j] = -(self.c/w\*\*2)\*(m\_temp[2,0]\*u-m\_temp[0,0]\*w)

#Y

self.Ae[i,j + 1] = -self.c/w\*\*2\*(m\_temp[2,1]\*u-m\_temp[0,1]\*w)

#Z

self.Ae[i,j + 2] = -self.c/w\*\*2\*(m\_temp[2,2]\*u-m\_temp[0,2]\*w)

#w

self.Ae[i,j + 3] = -self.c/w\*\*2\*((self.Y\_i - self.Y\_cj)\*(u\*m\_temp[2,2]-w\*m\_temp[0,2])

-(self.Z\_i - self.Z\_cj)\*(u\*m\_temp[2,1]-w\*m\_temp[0,1]))

#o

self.Ae[i,j + 4] = -self.c/w\*\*2\*((self.X\_i - self.X\_cj)\*(-w\*m.sin(self.o)\*m.cos(self.k)-u\*m.cos(self.o))

+(self.Y\_i - self.Y\_cj)\*(w\*m.sin(self.w)\*m.cos(self.o)\*m.cos(self.k)-u\*m.sin(self.w)\*m.sin(self.o))

+(self.Z\_i - self.Z\_cj)\*(-w\*m.cos(self.w)\*m.cos(self.o)\*m.cos(self.k)+u\*m.cos(self.w)\*m.sin(self.o)))

#k

self.Ae[i,j + 5] = -self.c\*v/w

#then odds (Y partial)

#X

self.Ae[i+1,j] = -self.c/w\*\*2\*(m\_temp[2,0]\*v-m\_temp[1,0]\*w)

#Y

self.Ae[i+1,j + 1] = -self.c/w\*\*2\*(m\_temp[2,1]\*v-m\_temp[1,1]\*w)

#Z

self.Ae[i+1,j + 2] = -self.c/w\*\*2\*(m\_temp[2,2]\*v-m\_temp[1,2]\*w)

#w

self.Ae[i+1,j + 3] = -self.c/w\*\*2\*((self.Y\_i - self.Y\_cj)\*(v\*m\_temp[2,2]-w\*m\_temp[1,2])

-(self.Z\_i - self.Z\_cj)\*(v\*m\_temp[2,1]-w\*m\_temp[1,1]))

#o

self.Ae[i+1,j + 4] = -self.c/w\*\*2\*((self.X\_i - self.X\_cj)\*(w\*m.sin(self.o)\*m.sin(self.k)-v\*m.cos(self.o))

+(self.Y\_i - self.Y\_cj)\*(-w\*m.sin(self.w)\*m.cos(self.o)\*m.sin(self.k)-v\*m.sin(self.w)\*m.sin(self.o))

+(self.Z\_i - self.Z\_cj)\*(w\*m.cos(self.w)\*m.cos(self.o)\*m.sin(self.k)+v\*m.cos(self.w)\*m.sin(self.o)))

#k

self.Ae[i+1,j + 5] = self.c\*u/w

## Tables.py

from scipy import misc

from scipy import stats

import pandas as pd

import numpy as np

class Tables():

"""

Parent class to PostAdjustmentTester which generates the significant values to increase modularity

"""

def \_\_init\_\_(self):

"""

"""

def newtons\_method(self, x, tolerance=0.0001):

while True:

x1 = x - self.f(x) / misc.derivative(self.f, x)

t = abs(x1 - x)

if t < tolerance:

break

x = x1

return x

def f(self, x):

return 1 - stats.chi2.cdf(x, self.r) - self.pvalue

def x\_2(self):

"""

Reference:

Code reformatted to return a single line of the desired x\_2 value based on our DOF (instead of a given value)

Code refers to functions "newtons\_method", "f", "x\_2"

https://moonbooks.org/Articles/How-to-create-a-Chi-square-table-using-python-/

Desc:

returns a chi-square dataframe row for the designated DOF

Input:

r: defrees of freedom

Output:

"""

self.pvalueList = [0.995, 0.99, 0.975, 0.95, 0.90, 0.10, 0.05, 0.025, 0.01, 0.005]

results = []

for i in range(self.r,self.r+1):

self.r = i

Result = []

for self.pvalue in self.pvalueList:

x0 = self.r # x0 approximation

x = self.newtons\_method(x0)

Result.append(x)

for i in range(10):

Result[i] = round(Result[i],3)

results.append(Result)

return pd.DataFrame(results, columns = self.pvalueList

## FileReader.py

import numpy as np

import pandas as pd

class File\_Reader():

"""

Contains a bunch of file reading functions so that the class may be imported when desired files what to be read in

"""

def \_\_init\_\_(self, tie\_file = 'engo531\_lab1.tie',

ext\_file = 'engo531\_lab1.ext',

int\_file = 'engo531\_lab1.int',

pho\_file = "engo531\_lab1.pho",

con\_file = "engo531\_lab1.con"

):

"""

Desc:

does not have any need to setup anything. More of just a function container

all id's are in strings

In:

Out:

self.tie: DF of tie points

self.ext: data frame of exterior orientation parameters

self.int: DF of interior orientation parameters

self.pho: Dataframe of image (photo) point obs

self.con: Df of control points

self.obj: control and tie point dataframes

"""

self.tie\_file = tie\_file

self.ext\_file = ext\_file

self.int\_file = int\_file

self.pho\_file = pho\_file

self.con\_file = con\_file

self.con = self.read\_con()

self.tie = self.read\_tie()

self.ext = self.read\_ext()

self.int = self.read\_int()

self.pho = self.read\_pho()

self.obs\_points()

def read\_tie(self):

"""

Desc:

Reads in the tie points as returns dataframe of the values

In:

filename, default set to lab1 filename

Out:

dataframe with columns "X, Y, Z" and index not set to point\_id

"""

df = pd.read\_csv(self.tie\_file, sep = "\t", header = None)

df.columns = ["point\_id", "X", "Y", "Z"]

#df = df.set\_index("point\_id")

#convert all value columns to flaots

df[["X", "Y", "Z"]] = df[["X", "Y", "Z"]].astype(float)

#convert ID's to strings

df[["point\_id"]] = df[["point\_id"]].astype(str)

#std for tie points is 1 pixel

return df

def read\_con(self):

"""

Desc:

Reads in the control points as returns dataframe of the values

In:

filename, default set to lab1 filename

Out:

dataframe with columns "X, Y, Z" and index not set to point\_id

"""

df = pd.read\_csv(self.con\_file, sep = "\t", header = None)

#cleaning the data

df = df.drop(4, axis=1)

df.columns = ["point\_id", "X", "Y", "Z"]

#df = df.set\_index("point\_id")

#convert all value columns to flaots

df[["X", "Y", "Z"]] = df[["X", "Y", "Z"]].applymap(np.float64)

#convert ID's to strings

df[["point\_id"]] = df[["point\_id"]].astype(str)

return df

def read\_ext(self):

"""

Desc:

Reads in the tie points as returns dataframe of the values

In:

filename, default set to lab1 filename

Out:

dataframe with columns "image\_id","camera\_id","Xc", "Yc", "Zc", "w", "o", "k" and index set to natural incrementation

"""

df = pd.read\_csv(self.ext\_file, sep = "\t", header = None)

#cleaning the data

df = df.drop([8,9,10,11,12,13,14], axis=1)

df.columns = ["image\_id","camera\_id","Xc", "Yc", "Zc", "w", "o", "k"]

#convert all value columns to flaots

#df = df[["Xc", "Yc", "Zc", "w", "o", "k"]].astype(float)

df[["Xc", "Yc", "Zc", "w", "o", "k"]] = df[["Xc", "Yc", "Zc", "w", "o", "k"]].applymap(np.float64)

#convert ID's to strings

df[["camera\_id"]] = df[["camera\_id"]].astype(str)

df[["image\_id"]] = df[["image\_id"]].astype(str)

return df

def read\_int(self):

"""

Desc:

Reads in the tie points as returns dataframe of the values

Currently only formatted for a single row. Multiple rows will need reformatting

In:

filename, default set to lab1 filename

Out:

dataframe with columns "camera\_id", 'xp', 'xp', "c" and index set to natural incrementation

"""

df = pd.read\_csv(self.int\_file, sep = "\t", header = None)

#cleaning the data

df = df.drop([0], axis=1)

#break column 2 into the proper X, Y, Z string

l = df.loc[0][2].split(" ")

l.remove('')

l = [x for x in l if x!='']

corrected = [df.loc[0][1]] + l

#recombine data again

df = pd.DataFrame([corrected], columns = ["camera\_id", 'xp', 'yp', "c"])

#convert ID's to strings

df[["camera\_id"]] = df[["camera\_id"]].astype(str)

#secure flaot type numbers

df[["c"]] = df[["c"]].astype(float)

df[["xp"]] = df[["xp"]].astype(float)

df[["yp"]] = df[["yp"]].astype(float)

return df

def read\_pho(self):

"""

Desc:

Reads in the pho (observation) points as returns dataframe of the values

Must have self.tie initialized

In:

self.tie

filename, default set to lab1 filename

Out:

dataframe with columns "point\_id", "image\_id", "x", "y" and index set to natural incrementation

"""

#uses mixed spacing to read in files... nbd ;-)

df = pd.read\_csv(self.pho\_file, header = None, delim\_whitespace =True)

#assign column values

df.columns = ["point\_id", "image\_id", "x", "y"]

#combines point\_id and image\_id for a unique identifier

df["unique\_id"] = df["point\_id"].to\_numpy()+df["image\_id"].astype(str).to\_numpy()

#convert ID's to strings

df[["point\_id"]] = df[["point\_id"]].astype(str)

df[["image\_id"]] = df[["image\_id"]].astype(str)

#sort values in ascending inage\_id's

df = df.sort\_values(by=['image\_id'])

#std for control points is .01mm and temporarily 10mm for tie

temp = []

for index, row in df.iterrows():

# print(row['point\_id'])

if any(self.tie["point\_id"] == row['point\_id']):

temp.append("u")

else:

temp.append('n')

df["knowns"] = temp

return df

def obs\_points(self):

"""

Desc:

Initializes the object point dataframe

\*\*\*may bee differentiating between tie points and control points\*\*\*

Input:

self.tie

self.con

Output:

self.obj

"""

self.obj = pd.concat([self.tie, self.con])

#convert ID's to strings

self.obj[["point\_id"]] = self.obj[["point\_id"]].astype(str)

## Bundle.py

from numpy import matrix as mat, matmul as mm

from numpy import transpose as t

import math as m

import numpy as np

import pandas as pd

from LeastSquares import LS

from FileReader import File\_Reader

class Bundle(LS, File\_Reader):

"""

Desc:

Contains the LS for all LSA info

Contains the Bundle for all Bundle Adjustment specific specs

"""

def \_\_init\_\_(self):

"""

Desc:

Input:

Output:

"""

LS.\_\_init\_\_(self)

File\_Reader.\_\_init\_\_(self)

self.initialize\_variables()

def initialize\_variables(self):

"""

Desc:

initializes import dimensions as taken in from the File\_Reader

Input:

Output:

self.ue

self.uo

self.n

"""

#pixel spacing (mm)

self.pix\_to\_m = 3.45e-3

#pixel spacing (mm)

self.delta\_x = 3.45e-6\*1000

self.delta\_y = 3.45e-6\*1000

#normal principal distance (mm)

self.n\_p\_d = 7

#number of pixels for total columns

self.Np = 3000

#number of rows of pixels

self.Mp = 4000

self.set\_ue()

self.set\_uo()

self.set\_n()

def set\_ue(self):

"""

Desc:

finds m from # of images and then makes ue = 6 \* m

Input:

self.ext

Output:

self.ue

"""

m = len(self.ext.index)

self.ue = 6 \* m

def set\_uo(self):

"""

Desc:

finds p from # of points (currently just tie) and then makes uo = 2 \* p

Input:

maybe self.con??

self.tie

Output:

self.uo

"""

#control stuff added

q = len(self.obj.index)

self.uo = 3 \* q

def set\_n(self):

"""

Desc:

finds n from total number of pixel observations

Input:

self.pho

Output:

self.n

"""

p = len(self.pho.index)

self.n = 2 \* p

def rhc(self, n\_ij = 2015.203, m\_ij = 1566.904):

"""

Desc:

converts from LHC to RHC

Must be formatted to assign or return the x, y coordinates as desired

Input:

n\_ij (number of columns for that pixel)

m\_ij (number of rows for that pixel)

Out:

self.x\_ij

self.y\_ij

"""

#self.x\_ij = (n\_ij-((self.Np/2)-.5))\*self.delta\_x

#self.y\_ij = (((self.Mp/2)-.5)-m\_ij)\*self.delta\_y

self.x\_ij = (n\_ij-((self.Np/2)-.5))\*self.delta\_x

self.y\_ij = (((self.Mp/2)-.5)-m\_ij)\*self.delta\_y

def M(self):

"""

Desc:

Generates the M rotation matrix (3x3)

converts from LHC to RHC

Must be formatted to assign or return the x, y coordinates as desired

Input:

w, in radians

k, in radians

o, in radians

Out:

none atm

"""

o = self.o

k = self.k

w = self.w

temp = mat(np.zeros((3,3)))

#row zero

temp[0,0] = m.cos(o)\*m.cos(k)

temp[0,1] = m.cos(w)\*m.sin(k)+m.sin(w)\*m.sin(o)\*m.cos(k)

temp[0,2] = m.sin(w)\*m.sin(k)-m.cos(w)\*m.sin(o)\*m.cos(k)

#row one

temp[1,0] = -m.cos(o)\*m.sin(k)

temp[1,1] = m.cos(w)\*m.cos(k)-m.sin(w)\*m.sin(o)\*m.sin(k)

temp[1,2] = m.sin(w)\*m.cos(k)+m.cos(w)\*m.sin(o)\*m.sin(k)

#row two

temp[2,0] = m.sin(o)

temp[2,1] = -m.sin(w)\*m.cos(o)

temp[2,2] = m.cos(w)\*m.cos(o)

#testing using matrix multiplication instead

#temp = mm(self.R3(k), mm(self.R2(o), self.R1(w)))

#for future reference

#w = m.atan(-temp[2,1]/temp[2,2])

#o = m.asin(temp[2,0])

#k = m.atan(-temp[2,1]/temp[0,0])

return temp

def U(self):

"""

Desc:

\*\*\*\*\*\*\*test values are for angles ATM\*\*\*\*\*\*\*\*\*\*\*\*

uses the angle values and input XYZ values to output U

Input:

w, in radians

k, in radians

o, in radians

X\_i,

self.X\_cj,

Y\_i,

self.Y\_cj,

self.Z\_i,

self.Z\_cj

Out:

none atm

"""

U = self.M()[0,0]\*(self.X\_i-self.X\_cj)+self.M()[0,1]\*(self.Y\_i-self.Y\_cj)+self.M()[0,2]\*(self.Z\_i-self.Z\_cj)

return U

def W(self):

"""

Desc:

\*\*\*\*\*\*\*test values are for angles ATM\*\*\*\*\*\*\*\*\*\*\*\*

uses the angle values and input XYZ values to output W

Input:

w, in radians

k, in radians

o, in radians

self.X\_i,

self.X\_cj,

self.Y\_i,

self.Y\_cj,

self.Z\_i,

self.Z\_cj

Out:

none atm

"""

W = self.M()[2,0]\*(self.X\_i-self.X\_cj)+self.M()[2,1]\*(self.Y\_i-self.Y\_cj)+self.M()[2,2]\*(self.Z\_i-self.Z\_cj)

return W

def V(self):

"""

Desc:

\*\*\*\*\*\*\*test values are for angles ATM\*\*\*\*\*\*\*\*\*\*\*\*

uses the angle values and input XYZ values to output W

Input:

w, in radians

k, in radians

o, in radians

self.X\_i,

self.X\_cj,

self.Y\_i,

self.Y\_cj,

self.Z\_i,

self.Z\_cj

Out:

none atm

"""

V = self.M()[1,0]\*(self.X\_i-self.X\_cj)+self.M()[1,1]\*(self.Y\_i-self.Y\_cj)+self.M()[1,2]\*(self.Z\_i-self.Z\_cj)

return V

def R1(self, o):

"""

Desc:

Returns R1 matrix

Input:

radians o

Output:

R1 (3x3)

"""

temp = mat(np.zeros((3,3)))

#row zero

temp[0,0] = 1

temp[0,1] = 0

temp[0,2] = 0

#row one

temp[1,0] = 0

temp[1,1] = m.cos(o)

temp[1,2] = m.sin(o)

#row two

temp[2,0] = 0

temp[2,1] = -m.sin(o)

temp[2,2] = m.cos(o)

return temp

def R2(self, o):

"""

Desc:

Returns R2 matrix

Input:

radians o

Output:

R2 (3x3)

"""

temp = mat(np.zeros((3,3)))

#row zero

temp[0,0] = m.cos(o)

temp[0,1] = 0

temp[0,2] = -m.sin(o)

#row one

temp[1,0] = 0

temp[1,1] = 1

temp[1,2] = 0

#row two

temp[2,0] = m.sin(o)

temp[2,1] = 0

temp[2,2] = m.cos(o)

return temp

def R3(self, o):

"""

Desc:

Returns R3 matrix

Input:

radians o

Output:

R3 (3x3)

"""

temp = mat(np.zeros((3,3)))

#row zero

temp[0,0] = -m.cos(o)

temp[0,1] = m.sin(o)

temp[0,2] = 0

#row one

temp[1,0] = -m.sin(o)

temp[1,1] = m.cos(o)

temp[1,2] = 0

#row two

temp[2,0] = 0

temp[2,1] = 0

temp[2,2] = 1

return temp